Second-Order Simple Grammars

Second-Order Simple Grammars

Colin Stirling

School of Informatics
University of Edinburgh
Edinburgh EH9 3JZ, UK
email: cps@inf.ed.ac.uk

1 Introduction

Higher-order notations for trees have a venerable history from the 1970s and 1980s when schemes (that is, functional programs without interpretations) and their relationship to formal language theory were first studied. Included are higher-order recursion schemes and pushdown automata. Automata and language theory study finitely presented mechanisms for generating languages. Instead of language generators, one can view them as process calculi, propagators of possibly infinite labelled transition systems. Recently, model-checking techniques have been successfully extended to these higher-order notations in the deterministic case [18, 9, 8, 21].

A long standing open question is: given two nth-order schemes do they generate the same tree? Courcelle [10] showed that for $n = 1$ the problem coincides with the language equivalence problem for deterministic pushdown automata (DPDA) that was subsequently solved positively by Sénizergues [23]. For $n > 1$, equivalence of *safe* nth-order recursion schemes coincides with equivalence between determinisitic nth-order pushdown automata [12, 18]. It is not known whether safety is a genuine restriction on expressive power: see [1].

Second-order pushdown automata involve finite-state control over a stack of stacks. They have applications in language theory as they characterize the *indexed* languages introduced by Aho [2]. Also, they generalize the "mildly" context-sensitive languages used in computational linguistics [29]. Aho defined these languages using indexed grammars and also characterized them in terms of nested stack automata [3]. Their characterization in terms of second-order pushdown automata is due to Maslov, who also defined a hierarchy of higher-order indexed languages characterized by higher-order pushdown automata, [20]. A more detailed account is given by Damm and Goerdt [12].

There has been considerable research activity on decision procedures for bisimulation equivalence between first-order systems, initiated with [4] for normed context-free grammars and then extended to classes of pushdown automata [26]. Recent results show that bisimulation equivalence is undecidable [17].

Here, we present a decidability result for equivalence of second-order systems. A configuration of a second-order pushdown automaton is a state and a stack of stacks. The operations pop stacks and push stacks onto it. We examine

deterministic second-order pushdown automata which generalize DPDA. A configuration of a DPDA is a state and a stack. Simple grammars are an instance of DPDA when there is a single state and no ϵ-transitions. So a configuration of a simple grammar is justa stack. Korenjak and Hopcroft showed that language equivalence is decidable between configurations of simple grammars [19]. Here, we introduce second-order simple grammars as the subset of second-order deterministic pushdown automata when there is a single state and no ϵ-transitions. A configuration of such a grammar is, therefore, a stack of stacks. We show that language equivalence is decidable for a subset of second-order simple grammars. The proof technique is based on bisimulation equivalence and some combinatorics about repetitions of stack extensions (loosely based on ideas from [28]). We view this result as a first step towards understanding the general equivalence problem for higher-order schemes.

In Section 2, we describe 2nd-order (deterministic) pushdown automata and in Section 3 we introduce 2nd-order simple grammars and the subset that we study. Some properties of the grammars are outlined in Section 4. In Sections 5 and 6 we present the equivalence decision procedure, using tableaux.

2 2nd-order pushdown automata

The following four finite sets are ingredients of a 2nd-order pushdown automaton, a 2PDA: states P, stack symbols S, alphabet A and basic transitions T. A basic transition is $pX \xrightarrow{a} q\theta$ where p and q are states in P, X is a stack symbol in S, $a \in \mathsf{A} \cup \{\epsilon\}$ and θ is an operation belonging to $\{\mathrm{swap}_\alpha, \mathrm{push}, \mathrm{pop} : \alpha \in \mathsf{S}^*\}$.

A 2-stack is a sequence of non-empty stacks $\beta_1 : \ldots : \beta_n$, so each $\beta_i \in \mathsf{S}^+$. We use ϵ for the empty stack and capital greek letters Γ, Δ, ... to range over sequences of stacks with Λ for the empty sequence. An operation θ is defined on a 2-stack as follows:

$$\begin{array}{ll} \mathrm{swap}_\alpha(X\beta : \Gamma) & = \alpha\beta : \Gamma \\ \mathrm{push}(\beta : \Gamma) & = \beta : \beta : \Gamma \\ \mathrm{pop}(\beta : \Gamma) & = \Gamma \end{array}$$

A configuration of a 2PDA consists of a state $p \in \mathsf{P}$ and a 2-stack Γ. The transitions of a configuration are defined by the following rule from the basic transitions T.

$$\text{PRE} \quad \text{If } pX \xrightarrow{a} q\theta \in \mathsf{T} \text{ then } pX\beta : \Gamma \xrightarrow{a} q\,\theta(X\beta : \Gamma)$$

A traditional automaton interpretation is that on input a with basic transition $pX \xrightarrow{a} q\theta$ the configuration $pX\beta : \Gamma$ in state p with X at the top of the first stack changes to state q and $\theta(X\beta : \Gamma)$ replaces $X\beta : \Gamma$. Alternatively, with respect to a generational or process calculus perspective the configuration $pX\beta : \Gamma$ generates, or performs, a and becomes $q\theta(X\beta : \Gamma)$. In both accounts ϵ-transitions have a special status. If $a = \epsilon$ then the configuration may change

without reading an input or it may become $q\theta(X\beta : \Gamma)$ silently without performing an observable action. In the following we abbreviate a basic transition $pX \xrightarrow{a} q\,\mathrm{swap}_\alpha$ to $pX \xrightarrow{a} q\alpha$.

The *transition graph* $\mathsf{G}(p\Gamma)$ is generated by deriving all possible transitions from $p\Gamma$ and every configuration reachable from it using the rule PRE.

Example 1. Consider the following basic transitions.

$$\begin{array}{llll} pZ \xrightarrow{a} qZ & qZ \xrightarrow{a} qAZ & qA \xrightarrow{a} qAA & qA \xrightarrow{b} r\,\mathrm{push} \\ rA \xrightarrow{b} r\epsilon & rZ \xrightarrow{c} s\,\mathrm{pop} & sA \xrightarrow{c} s\epsilon & sZ \xrightarrow{\epsilon} s\,\mathrm{pop} \end{array}$$

Part of the transition graph $\mathsf{G}(pZ)$ is depicted in Figure 1. □

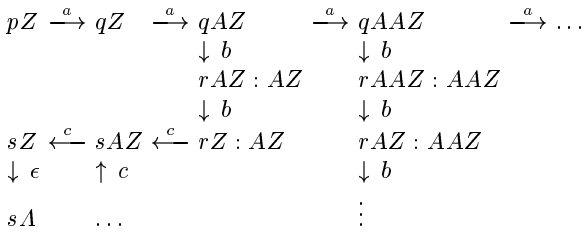

Fig. 1. A 2PDA

A 2PDA is presentable in normal form, up to isomorphism of transition graphs, where each transition of the form $pX \xrightarrow{a} q\alpha \in \mathsf{T}$ obeys the constraint that the length of α, $|\alpha|$, is at most 2. Enforcement of the normal form is easy to achieve, by introducing extra stack symbols.

Definition 1. *The* language *of a configuration $p\Delta$, $L(p\Delta)$, is the set of words $w \in \mathsf{A}^*$ such that $p\Delta \xrightarrow{w} q\Lambda$ for some q.*

When recognising any such word the 2-stack is thereby emptied. For instance, $L(pZ)$ in the case of Example 1 is $\{a^n b^n c^n : n \geq 2\}$ which is a context-sensitive language. This is called *empty stack acceptance*. A word $w \in \mathsf{A}^*$ is in $L(p\Delta)$ if there is a w-path from $p\Delta$ to a terminal state $q\Lambda$ for some q in the graph $\mathsf{G}(p\Delta)$. The languages recognized coincide with those recognized if final states were also included.

Our definition of a 2PDA is based on [18] except that it explicitly extends a standard PDA (because of swap transitions). It is simpler than Maslov's, Damm and Goerdt's definition [20, 12]. In their case, a 2-stack is a sequence of pairs (X_i, α_i) where $X_i \in \mathsf{S}$, with operations pop_1, pop_2, $\mathrm{push}_1(\alpha)$, $\mathrm{push}_2(\alpha)$ which work as follows: $\mathrm{pop}_1[(X, \alpha_1) : \Gamma] = \Gamma$, $\mathrm{pop}_2[(X, Y\alpha) : \Gamma] = (X, \alpha) : \Gamma$, $\mathrm{push}_1(Z_1 Z_2)[(X, \alpha) : \Gamma] = (Z_1, \alpha) : (Z_2, \alpha) : \Gamma$ and $\mathrm{push}_2(Z_1 Z_2)[(X, \alpha) : \Gamma]$

$= (X, Z_1 Z_2 \alpha) : \Gamma$. There is no loss in expressive power (with respect to language equivalence) as these operations can be simulated by families of 2PDA operations.

The family of languages recognized by 2PDA is the *indexed languages*, introduced by Aho in 1968 [2, 3], which permit some context-dependency, as Example 1 illustrates. Aho offers a grammatical method for generating them as well as an automata theoretic method (using nested stack automata) which turns out to be equivalent to the 2PDA, as shown by Maslov [20]. An equivalent, schema-like, formalism is the OI macro-grammars of Fischer [14]. Aho also shows that the indexed languages are context-sensitive which is not obvious because repeated push transitions can increase the size of a configuration non-linearly. They form an AFL and are a proper subset of the context-sensitive languages: $\{(ab^n)^n : n \geq 0\}$ is not an indexed language via a pumping lemma for them [16, 5]. The subset of *linear* indexed languages is the mildly context-sensitive languages generated by tree adjoining grammars [29].

A 2PDA is *deterministic* if T obeys the following conditions.

- if $pX \xrightarrow{a} q\theta$ and $pX \xrightarrow{a} r\lambda$ then $q = r$ and $\theta = \lambda$
- if $pX \xrightarrow{\epsilon} q\theta$ and $pX \xrightarrow{a} r\lambda$ then $a = \epsilon$

Example 1 is a determinisitic 2PDA. The equivalence question, whether two configurations of a determinisitic 2PDA recognise the same language, generalizes the DPDA equivalence problem, that was solved positively by Sénizergues [25, 23, 24, 27, 28]. A DPDA configuration $p\alpha$ can be coded as a deterministic 2PDA configuration $p\alpha Z$ where Z is a new end of stack marker with the extra transitions $qZ \xrightarrow{\epsilon} q$ pop for each state q.

Due to empty stack acceptance, the language recognized by a deterministic 2PDA has the prefix free property: if $w \in L(p\Delta)$ then no proper prefix v of w can belong to $L(p\Delta)$. However, as with DPDA and empty stack acceptance, for any deterministic indexed language L, when defined in the Maslov style [22] with final state acceptance, there is a deterministic 2PDA that accepts $\{w\$: w \in L\}$ where $\$$ is a new alphabet symbol: deterministic 2PDA coincide with deterministic Maslov pushdown automata with empty stack acceptance. The deterministic indexed languages are closed under complement (and are therefore a proper subset of the indexed languages) and include inherently ambiguous context-free languages such as $\{a^i b^j c^k : i, j, k > 0 \text{ and } i = j \text{ or } j = k\}$ [22].

3 Second-order simple grammars

In this section we consider second-order simple grammars, 2SGs. These are determinisitic 2PDAs which have just one state and no ϵ-transitions. We can therefore drop the state from transitions and configurations: transitions now have the form $X \xrightarrow{a} \theta$ and a configuration has the form Δ. Reachability properties of their nondeterministic version, at higher-orders, have been examined in [6]. We conjecture that simple grammars defined from Maslov pushdown automata are more expressive than 2SGs.

The DPDA correlate of 2SGs are *simple grammars*. A simple grammar contains basic deterministic transitions $X \xrightarrow{a} \alpha$, $a \in \mathsf{A}$, and the language of a configuration β, $L(\beta)$, is the set $\{w : \beta \xrightarrow{w} \epsilon\}$. Decidability of language equivalence between two configurations of a simple grammar was shown by Korenjak and Hopcroft [19]. However, language containment is undecidable [15].

It is unclear if there are alternative characterizations of 2SGs in terms of subsets of schema or macro-grammars. The restriction to a single state suggests that we should examine their monadic versions. We leave this for further work. The following example illustrates that there are interesting 2SGs.

Example 2. Consider the following 2SG

$$A \xrightarrow{a} AA \quad A \xrightarrow{b} \text{push} \quad A \xrightarrow{c} \epsilon \quad Z \xrightarrow{c} \text{pop}$$

Part of the graph $\mathsf{G}(AZ)$ is depicted in Figure 2. $L(AZ) \cap a^*b^*c^*$ is the language

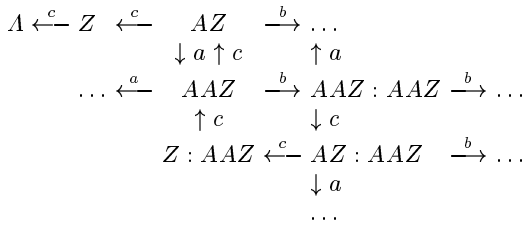

Fig. 2. A 2SG

$\{a^n b^k c^{(k+1)(n+2)} : n, k \geq 0\}$ which is not context-free by the pumping lemma for context-free languages. Therefore, $L(AZ)$ is also not context-free. Consequently, 2SGs are strictly more expressive than simple grammars. Also, they are not subsumed by pushdown automata. □

Example 3. 2SGs even without push transitions can be complex.

$$X \xrightarrow{a} YX \quad X \xrightarrow{b} \epsilon \quad Y \xrightarrow{b} X \quad Y \xrightarrow{c} Z \quad Z \xrightarrow{b} U \quad U \xrightarrow{b} \text{pop}$$
$$A \xrightarrow{a} C \quad A \xrightarrow{b} \epsilon \quad C \xrightarrow{b} AA \quad C \xrightarrow{c} W \quad W \xrightarrow{b} \text{pop}$$

Here, $L(XZ) = L(AW : W)$. The graph $\mathsf{G}(XZ)$ involves infinite indegree because $UX^nZ \xrightarrow{b} \Lambda$ for any n. □

Definition 2. *For each stack symbol X, let $\Lambda(X)$ be the length of a shortest word w, if it exists, such that $X \xrightarrow{w} \Lambda$, $\epsilon(X)$ be the length of a shortest word w, if it exists, such that $X \xrightarrow{w} \epsilon$ and $P(X)$ be the length of a shortest word wa, if it exists, such that $X \xrightarrow{w} Z\alpha$ and $Z \xrightarrow{a} \text{push} \in \mathsf{T}$.*

It is easy to compute whether $\Lambda(X)$, $\epsilon(X)$ or $P(X)$ are defined, and what their values are when defined. First we start by computing the cases of length 1: there must be basic transitions $X \xrightarrow{a}$ pop, $X \xrightarrow{a} \epsilon$ or $X \xrightarrow{a}$ push. To check for length n, we examine basic transitions $X \xrightarrow{a} W$ and $X \xrightarrow{a} YZ$: if $\epsilon(X)$ is not yet defined, and $\epsilon(W) = n-1$ or $\epsilon(Y) + \epsilon(Z) = n-1$ then $\epsilon(X) = n$; if $\Lambda(X)$ is not yet defined and $\Lambda(W) = n-1$ or $\Lambda(Y) = n-1$ or $\epsilon(Y) + \Lambda(Z) = n-1$ then $\Lambda(X) = n$; and, similarly, for $P(X)$ when it is currently undefined. The iteration stops at the first length $2k+1$ such that no $\Lambda(X)$, $\epsilon(X)$ or $P(X)$ has length more than k. At this point, any remaining $\Lambda(X)$, $\epsilon(X)$ and $P(X)$ are undefined. Clearly, no $\Lambda(X)$, $\epsilon(X)$, $P(X)$ can exceed $2^{|\mathsf{S}|}$. In the case of Example 2, $\Lambda(A)$, $\epsilon(Z)$ and $P(Z)$ are not defined and $\Lambda(Z)$, $\epsilon(A)$ and $P(A)$ are all 1. In Example 3, $\Lambda(X) = 4$, $\Lambda(Y) = 3$ and $\Lambda(A) = 3$.

Definition 3. *A 2SG is* special *if for each X, $\Lambda(X)$ or $P(X)$ is defined.*

The 2SGs in Examples 2 and 3 are special. We now state the main result of the paper.

Theorem 1. *If Γ, Δ are configurations of a special 2SG then it is decidable whether $L(\Gamma) = L(\Delta)$.*

The result strictly generalizes the equivalence problem for simple grammars. Consider a simple grammar with basic transitions of the form $X \xrightarrow{a} \alpha$. We transform it into a special 2SG. First, we extend the alphabet with two new symbols \$, # and add an end of stack marker Z with basic transition $Z \xrightarrow{\$}$ pop. For each stack symbol X we also add the transition $X \xrightarrow{\#}$ push. For any two configurations α and β of the simple grammar, $L(\alpha) = L(\beta)$ iff $L(\alpha Z) = L(\beta Z)$ in the transformed 2SG.

4 Some properties of special 2SGs

We quickly consider why language equivalence is decidable for simple grammars. A stack symbol X is *normed* if $\epsilon(X)$ is defined. Clearly, $L(\alpha) = \emptyset$ iff α contains an unnormed stack symbol. So we can put a simple grammar into normal form where all stack symbols are normed. With this assumption language equivalence coincides with bisimulation equivalence because of determinism and normedness. We write $\alpha \sim \beta$ if $L(\alpha) = L(\beta)$.

Proposition 1. *$\alpha\delta \sim \beta\delta$ iff $\alpha \sim \beta$ iff $\delta\alpha \sim \delta\beta$.*

Decidability of equivalence now follows reasonably straightforwardly via decomposition and substitutivity: for instance, if $X\alpha \sim \beta\delta$ and $\alpha \sim \beta'\delta$ then $X\beta' \sim \beta$. Decomposition can be extended to *unique* prime decomposition, see [7] for details.

In the case of 2SGs there are *two* notions of stack composition: one between stacks and the other within a stack. Again, we can easily check if a configuration $L(\Gamma) = \emptyset$ using the definitions of $\Lambda(X)$ and $\epsilon(X)$ from the previous section. Proposition 1 generalizes to composition between stacks for arbitrary 2SGs.

Proposition 2. *Assume $L(\Gamma)$, $L(\Sigma)$ and $L(\Delta)$ are all nonempty. It follows that $L(\Gamma : \Delta) = L(\Sigma : \Delta)$ iff $L(\Gamma) = L(\Sigma)$ iff $L(\Delta : \Gamma) = L(\Delta : \Sigma)$.*

Proof. Assume $L(\Gamma)$, $L(\Sigma)$, $L(\Delta)$ are nonempty and $L(\Gamma : \Delta) = L(\Sigma : \Delta)$. If $w \in L(\Gamma : \Delta)$ then $w = w_1 w_2$ and $w_1 \in L(\Gamma)$ and $w_2 \in L(\Delta)$. Let v be a shortest word in $L(\Delta)$. If $w_1 \notin L(\Sigma)$ then there are two cases. First, a proper prefix w_{11} of w_1 is in $L(\Sigma)$. It follows that $w_{11}v \in L(\Sigma : \Delta)$ and $w_{11}v \notin L(\Gamma : \Delta)$ which is a contradiction. Secondly, $w_1 w_{21} \in L(\Sigma)$ where $w_2 = w_{21}w_{22}$ and $w_{21} \neq \epsilon$. Therefore, $w_1 v \in L(\Gamma : \Delta)$ and $w_1 v \notin L(\Sigma : \Delta)$ which again is a contradiction. Arguments for all the other cases are similar. □

However, there are not the same properties for composition within a stack. It is possible for $L(\alpha) = L(\beta)$ and $L(\alpha\delta) \neq L(\beta\delta)$ and for $L(\alpha\delta) = L(\beta\delta)$ and $L(\alpha) \neq L(\beta)$. A simple case is $X \xrightarrow{a} \epsilon$ and $X \xrightarrow{b}$ pop and $Y \xrightarrow{b}$ pop. Although $L(X) = L(Y)$, $L(XY) \neq L(YY)$ because of the disitinguishing word ab.

We introduce an extra configuration $\emptyset$ with $L(\emptyset) = \emptyset$. In the following we always assume that when we write a configuration $\Gamma \neq \emptyset$ then $L(\Gamma) \neq \emptyset$. We define the operation $\Gamma \cdot a$ as follows for $a \in \mathsf{A}$.

Definition 4. *If $\Gamma \xrightarrow{a} \Gamma'$ and $L(\Gamma') \neq \emptyset$ then $\Gamma \cdot a = \Gamma'$ otherwise $\Gamma \cdot a = \emptyset$.*

Proposition 3. *If $X \xrightarrow{a}$ push $\in \mathsf{T}$ then $((X\alpha : \Gamma) \cdot a) = X\alpha : X\alpha : \Gamma$.*

We extend Definition 4 to words.

Definition 5. *$\Gamma \cdot \epsilon = \Gamma$ and $\Gamma \cdot aw = (\Gamma \cdot a) \cdot w$.*

We now come to a key, perhaps surprizing, property of a special 2SG which is essential to the decidability proof.

Proposition 4. *Assume $L(X\alpha : \Gamma) = L(Y\beta : \Delta)$ for configurations of a special 2SG. If $X \xrightarrow{a}$ push $\in \mathsf{T}$ then $Y \xrightarrow{a}$ push $\in \mathsf{T}$ and $L(X\alpha) = L(Y\beta)$.*

Proof. Suppose $L(X\alpha : \Gamma) = L(Y\beta : \Delta)$ and $X \xrightarrow{a}$ push $\in \mathsf{T}$. By assumption $L(X\alpha : \Gamma) \neq \emptyset$. If $Y \xrightarrow{a}$ push $\notin \mathsf{T}$ then $Y \xrightarrow{a} \theta$ and $\theta =$ pop or swap$_{\gamma_1}$. Consider the case $\theta =$ pop. Therefore, $L(X\alpha : X\alpha : \Gamma) = L(\Delta)$. But then by Proposition 2, $L(X\alpha : X\alpha : \Gamma) = L(X\alpha : Y\beta : \Delta) = L(\Delta)$ which is a contradiction. Consequently, $\theta =$ swap$_{\gamma_1}$ and $L(X\alpha : X\alpha : \Gamma) = L(\beta_1 : \Delta)$ where $\theta(Y\beta) = \beta_1$. Now we repeat the argument for Y_1 which is the head stack symbol of β_1. We show that $Y_1 \xrightarrow{a}$ push $\notin \mathsf{T}$. Assume it is. By Proposition 2, $L(X\alpha : X\alpha : X\alpha : \Gamma) = L(X\alpha : \beta_1 : \Delta) = L(\beta_1 : \beta_1 : \Delta)$ and so $L(X\alpha) = L(\beta_1)$. But $L(X\alpha : X\alpha : \Gamma) = L(\beta_1 : Y\beta : \Delta) = L(\beta_1 : \Delta)$ which is a contradiction. Therefore, $Y_1 \xrightarrow{a} \theta_1$ and $\theta_1 =$ pop or swap$_{\gamma_2}$. The argument above shows that $\theta_1 \neq$ pop. Therefore, $L(X\alpha : X\alpha : X\alpha : \Gamma) = L(\beta_2 : \Delta)$ where $\beta_2 = \theta_1(\beta_1)$. Now, we repeat the argument for Y_2 which is the head of β_2. Again, $X \xrightarrow{a}$ push $\in \mathsf{T}$ and by the arguments above $Y_2 \xrightarrow{a}$ swap$_{\gamma_2}$. After n steps, we have $L((X\alpha)^{n+1} : \Gamma) = L(\beta_n : \Delta)$. As $\Lambda(X\alpha) > 0$, it follows that

$\Lambda(\beta_n) = \Lambda(\beta_{n-1}) + \Lambda(X\alpha)$: we now use this property to obtain a contradiction when the 2SG is special. Let $n > 2 \times 2^{|\mathsf{S}|}$. Consider Y_n the head variable of β_n. As the 2SG is special, $\Lambda(Y_n)$ or $P(Y_n)$ is defined. Assume the first, and let w be a shortest word such that $Y_n \xrightarrow{w} \Lambda$. It follows that $L(((X\alpha)^{n+1} : \Gamma) \cdot w) = L(\Lambda)$ which is a contradiction. Similarly, if w is a shortest word that $Y_n \xrightarrow{w}$ push then $\beta_n \cdot w = \beta_{n+1} : \beta_{n+1}$. However, $\Lambda(\beta_{n+1}) > 2^{|\mathsf{S}|}$ which contradicts that $L(((X\alpha)^{n+1} : \Gamma) \cdot w) = L(\beta_{n+1} : \beta_{n+1} : \Delta)$. □

We introduce non-standard bisimulation approximants.

Definition 6. *We define $\sim_n$, $n \geq 0$, iteratively as follows.*

1. *$\Gamma \sim_0 \Delta$ iff $\Gamma = \emptyset = \Delta$ or $\Gamma \neq \emptyset$ and $\Delta \neq \emptyset$.*
2. *$\Lambda \sim_{n+1} \Lambda$ and $\emptyset \sim_{n+1} \emptyset$*
3. *$X\alpha : \Gamma \sim_{n+1} Y\beta : \Delta$ just in case*
   (a) *$\Lambda(X\alpha : \Gamma) = \Lambda(Y\beta : \Delta)$*
   (b) *$X \xrightarrow{a}$ push iff $Y \xrightarrow{a}$ push, and*
   (c) *for each $a \in \mathsf{A}$, $(X\alpha : \Gamma) \cdot a \sim_n (Y\beta : \Delta) \cdot a$.*

Built into this definition is the idea that an immediate bisimulation error occurs if configurations do not agree on length of their shortest words or if push actions are not matched. These non-standard approximants will be critical to the decidability proof later. We write $\Gamma \sim \Delta$ if for all n, $\Gamma \sim_n \Delta$.

Proposition 5. 1. *$L(\Gamma) = L(\Delta)$ iff $\Gamma \sim \Delta$.*
2. *If $\Gamma \sim_n \Delta$ and $\Delta \sim_n \Sigma$ then $\Gamma \sim_n \Sigma$.*
3. *If $\Gamma \not\sim_n \Delta$ and $\Delta \sim_{n+k} \Sigma$ then $\Gamma \not\sim_n \Sigma$.*

5 Tableaux

The decision procedure for special 2SGs is a *tableau proof system*, consisting of proof rules which allow goals to be reduced to subgoals. Goals and subgoals are all of the form $\Gamma \doteq \Delta$, "is $\Gamma \sim \Delta$?", where Γ and Δ are configurations of a special 2SG. The tableau proof rules are contained in Figure 3.

The initial tableau proof rule is UNF (unfold). The goal, $\Gamma \doteq \Delta$ reduces to the subgoals $(\Gamma \cdot a) \doteq (\Delta \cdot a)$ for each $a \in \mathsf{A}$. The application of this simple rule is both "complete" and "sound". Completeness is the property that if the goal, $\Gamma \doteq \Delta$, is true then so are all the subgoals, $(\Gamma \cdot a_i) \doteq (\Delta \cdot a_i)$.

Proposition 6. *If $\Gamma \sim \Delta$, then for all $a \in \mathsf{A}$, $(\Gamma \cdot a) \sim (\Delta \cdot a)$.*

Soundness is the converse, that if all the subgoals are true then so is the goal which is equivalent to, if the goal is false, $\Gamma \not\sim \Delta$, then so is at least one of the subgoals. However, there is a finer account that uses approximants. We assume that, at least, $\Gamma \sim_1 \Delta$ (so push transitions have to be matched).

Proposition 7. *If $\Gamma \sim_{n+1} \Delta$ and $\Gamma \not\sim_{n+2} \Delta$, then $(\Gamma \cdot a) \not\sim_{n+1} (\Delta \cdot a)$ for some $a \in \mathsf{A}$.*

UNF

$$\frac{\Gamma \doteq \Delta}{(\Gamma \cdot a_1) \doteq (\Delta \cdot a_1) \ \ldots \ (\Gamma \cdot a_k) \doteq (\Delta \cdot a_k)} \ \mathsf{A} = \{a_1, \ldots, a_k\}$$

SIMP(L) and SIMP(R)

$$\frac{\alpha X \alpha' : \Gamma \doteq \Delta}{\alpha X : \Gamma \doteq \Delta} \ \epsilon(X) \text{ undefined} \qquad \frac{\Delta \doteq \alpha X \alpha' : \Gamma}{\Delta \doteq \alpha X : \Gamma} \ \epsilon(X) \text{ undefined}$$

DEC(L) and DEC(R)

$$\frac{\alpha : \Gamma \doteq \beta : \Delta}{\alpha : (\beta \cdot w) \doteq \beta \quad \Gamma \doteq (\beta \cdot w) : \Delta} \ C \qquad \frac{\beta : \Delta \doteq \alpha : \Gamma}{\beta \doteq \alpha : (\beta \cdot w) \quad (\beta \cdot w) : \Delta \doteq \Gamma} \ C$$

where C is the condition

1. $\Lambda(\alpha) \leq \Lambda(\beta)$ and $\Delta \neq \Lambda$
2. w is a smallest word such that $\alpha \xrightarrow{w} \Lambda$
3. $(\beta \cdot w) \neq \emptyset$

Fig. 3. Tableau proof rules

The second rules are SIMP (simplification) that reduce goals. If $\epsilon(X)$ is not defined then $\alpha X \alpha'$ can be reduced to αX. The following implies soundness and completeness of SIMP.

Proposition 8. *If $\epsilon(X)$ is undefined then for all n and Γ $\alpha X \alpha' : \Gamma \sim_n \alpha X : \Gamma$.*

The final rules are DEC for decomposition. We only decompose $\alpha : \Gamma = \beta : \Delta$ when Δ is non-empty. The following capture completeness and soundness.

Proposition 9. *Assume $\Lambda(\alpha) \leq \Lambda(\beta)$, w is a smallest word such that $\alpha \xrightarrow{w} \Lambda$ and $(\beta \cdot w) \neq \emptyset$.*

1. If $\alpha : \Gamma \sim \beta : \Delta$, then $\alpha : (\beta \cdot w) \sim \beta$ and $\Gamma \sim (\beta \cdot w) : \Delta$.
2. If $\alpha : \Gamma \not\sim_n \beta : \Delta$ then $\alpha : (\beta \cdot w) \not\sim_n \beta$ or $n > |w|$ and $\Gamma \not\sim_{n-|w|} (\beta \cdot w) : \Delta$.

Example 4. The following is an application of DEC(R) to a goal whose 2SG is Example 3.

$$\frac{XXXZ : XZ \doteq AAAW : AW : W}{XXXZ \doteq AAAW : UXXXZ \quad UXXXZ : XZ \doteq AW : W}$$

Here, $AAAW \xrightarrow{acb} \Lambda$ and $UXXXZ = (XXXZ \cdot acb)$. □

6 Successful tableaux

In the previous section we presented and justified tableau proof rules. We now show that these rules lead to an effective decision procedure for checking equivalence of configurations of special 2SGs. A missing ingredient in the tableau

description is when a current goal is final. The tableau procedure starts with an initial goal, $\Gamma \doteq \Delta$, "is $\Gamma \sim \Delta$?", and one then builds a proof tree by applying the tableau rules. Goals are thereby reduced to subgoals. Rules are not applied to final goals.

A 2SG is deterministic, and therefore we would prefer that there is just one tableau proof tree for any starting goal. To achieve uniqueness of tableau, we assume a linear ordering on the alphabet A. This ordering is used in an application of UNF, so the subgoals are ordered relative to this ordering. It is also used in the DEC rules to define a unique smallest word such that $\alpha \xrightarrow{w} \Lambda$: if there is more than one word of the same length with this property, we choose amongst them the word that is lexicographically least with repect to the ordering on A. In the case of the SIMP rules we assume that $\epsilon(\alpha)$ is defined: we always try to find the first stack symbol X in the initial stack such that $\epsilon(X)$ is undefined.

Next, we assume that the tableau proof rules are applied in the following order: DEC(L), DEC(R), SIMP(L), SIMP(R), UNF. Given a goal one tries first to apply DEC(L), and if it is not applicable then one tries DEC(R), and so on. A tableau proof tree is built breadth first starting with leftmost non-final goals.

Example 5. Here is part of the tableau proof tree for the goal $XZ \doteq AW : W$ whose 2SG is Example 3.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{XZ \doteq AW : W}{YXZ \doteq CW : W}\,\text{UNF} \quad \cdots}{XXZ \doteq AAW : W}\,\text{UNF}}{YXXZ \doteq CAW : W}\,\text{UNF} \quad \cdots}{\cdots} \qquad \dfrac{\dfrac{ZXZ \doteq WW : W}{Z \doteq W : W}\,\text{SIMP(L)}}{\cdots}\,\text{SIMP(R)}}{}$$

Here we have missed out subgoals of the form $\emptyset \doteq \emptyset$. There is an application of SIMP(L) to $ZXZ \doteq WW : W$ because $\epsilon(Z)$ is not defined. □

To show decidability we intend to show that associated with any starting goal $\Gamma \doteq \Delta$ is a unique boundedly *finite* proof tree. However, in Example 5 there appears to be the following potentially infinite branch of goals.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{XZ \doteq AW : W}{YXZ \doteq CW : W}}{XXZ \doteq AAW : W}}{YXXZ \doteq CAW : W}}{XXXZ \doteq AAAW : W}}{\cdots}$$

This will be dealt with by the definition of *final* goal.

Final goals are either *unsuccessful* or *successful*. There is just one kind of unsuccessful goal: $\Gamma \doteq \Delta$ where $\Gamma \not\sim_1 \Delta$. For successful final goals, first we include the identity, $\Gamma \doteq \Gamma$, which is clearly true. However, there is another kind based on repeating patterns of stack extensions (inspired by the extension theorem in [28] which was generalized to the subwords lemma in [25]).

We are interested in goals $\alpha : \Gamma \doteq \beta$ or $\beta \doteq \alpha : \Gamma$ where one side consists of a single stack only: application of the DEC proof rules yield such subgoals. Given a goal $\alpha\alpha_1 : \Gamma \doteq \beta\beta_1$, where α and β are not ϵ, we say that $\alpha\gamma_1\alpha_1 : \Gamma \doteq \beta\lambda_1\beta_1$ is an (γ_1, λ_1)-*extension* of it and (γ_1, λ_1) is the *extension*. We now come to the key property that will limit the size of a proof tree.

Proposition 10. *If α and β are not ϵ and*

(1) $\alpha\alpha_1 : \Gamma \sim_n \beta\beta_1$ and (5) $\alpha\alpha_2 : \Gamma \sim_n \beta\beta_2$
(2) $\alpha\gamma_1\alpha_1 : \Gamma \sim_n \beta\lambda_1\beta_1$ and (6) $\alpha\gamma_1\alpha_2 : \Gamma \sim_n \beta\lambda_1\beta_2$
(3) $\alpha\gamma_2\gamma_1\alpha_1 : \Gamma \sim_n \beta\lambda_2\lambda_1\beta_1$ and (7) $\alpha\gamma_2\gamma_1\alpha_2 : \Gamma \sim_n \beta\lambda_2\lambda_1\beta_2$
(4) $\alpha\gamma_1\gamma_2\gamma_1\alpha_1 : \Gamma \sim_n \beta\lambda_1\lambda_2\lambda_1\beta_1$

then (8) $\alpha\gamma_1\gamma_2\gamma_1\alpha_2 : \Gamma \sim_n \beta\lambda_1\lambda_2\lambda_1\beta_2$.

Proof. Assume (1) – (7) but (8) is false. So, $\alpha\gamma_1\gamma_2\gamma_1\alpha_2 : \Gamma \not\sim_n \beta\lambda_1\lambda_2\lambda_1\beta_2$. Because of (1) – (7), the bisimulation error in (8) cannot be caused by the heads α and β. Therefore, by repeated application of Proposition 7 there is a w such that one of the following hold. (An easy argument shows that w cannot involve a push transition.)

A) $\alpha \cdot w = \Lambda$, $\beta \cdot w$ is defined and $\Gamma \not\sim_{n-|w|} (\beta \cdot w)\lambda_1\lambda_2\lambda_1\beta_2$.
B) $\alpha \cdot w = \epsilon$, $\beta \cdot w$ is defined and $\gamma_1\gamma_2\gamma_1\alpha_2 : \Gamma \not\sim_{n-|w|} (\beta \cdot w)\lambda_1\lambda_2\lambda_1\beta_2$.
C) $\beta \cdot w = \epsilon$, $\alpha \cdot w$ is defined and $(\alpha \cdot w)\gamma_1\gamma_2\gamma_1\alpha_2 : \Gamma \not\sim_{n-|w|} \lambda_1\lambda_2\lambda_1\beta_2$.

Consider B): the others are similar. Because of (1) – (7) we know that

(11) $\alpha_1 : \Gamma \sim_{n-|w|} (\beta \cdot w)\beta_1$ (51) $\alpha_2 : \Gamma \sim_{n-|w|} (\beta \cdot w)\beta_2$
(21) $\gamma_1\alpha_1 : \Gamma \sim_{n-|w|} (\beta \cdot w)\lambda_1\beta_1$ (61) $\gamma_1\alpha_2 : \Gamma \sim_{n-|w|} (\beta \cdot w)\lambda_1\beta_2$
(31) $\gamma_2\gamma_1\alpha_1 : \Gamma \sim_{n-|w|} (\beta \cdot w)\lambda_2\lambda_1\beta_1$ (71) $\gamma_2\gamma_1\alpha_2 : \Gamma \sim_{n-|w|} (\beta \cdot w)\lambda_2\lambda_1\beta_2$
(41) $\gamma_1\gamma_2\gamma_1\alpha_1 : \Gamma \sim_{n-|w|} (\beta \cdot w)\lambda_1\lambda_2\lambda_1\beta_1$

We now consider $\alpha \cdot w = \epsilon$, $\beta \cdot w$ is defined and $\gamma_1\gamma_2\gamma_1\alpha_2 : \Gamma \not\sim_{n-|w|} (\beta \cdot w)\lambda_1\lambda_2\lambda_1\beta_2$ and (21), (41) and (61). There are two cases depending on whether $(\beta \cdot w) = \epsilon$. Assume it is not. The bisimulation error cannot be caused by the heads γ_1 and $(\beta \cdot w)$. Therefore there is a word w_1 such that one of the following hold.

BA) $\gamma_1 \cdot w_1 = \Lambda$, $\beta \cdot ww_1$ is defined and $\Gamma \not\sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\lambda_2\lambda_1\beta_2$.
BB) $\gamma_1 \cdot w_1 = \epsilon$, $\beta \cdot ww_1$ is defined and $\gamma_2\gamma_1\alpha_2 : \Gamma \not\sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\lambda_2\lambda_1\beta_2$.
BC) $\beta \cdot ww_1 = \epsilon$, $\gamma_1 \cdot w_1$ is defined and $(\gamma_1 \cdot w_1)\gamma_2\gamma_1\alpha_2 : \Gamma \not\sim_{n-|ww_1|} \lambda_1\lambda_2\lambda_1\beta_2$.

In the case of BA) we also know

$$(211)\ \Gamma \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\beta_1$$
$$(411)\ \Gamma \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\lambda_2\lambda_1\beta_1$$
$$(611)\ \Gamma \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\beta_2$$

Thus, we now get a contradiction using these because from Proposition 5

$$(\beta \cdot ww_1)\lambda_1\lambda_2\lambda_1\beta_1 \not\sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\lambda_2\lambda_1\beta_2$$
$$(\beta \cdot ww_1)\lambda_1\beta_1 \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\beta_2$$

In the case of BB) we also know that

$$(212)\ \alpha_1 : \Gamma \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\beta_1$$
$$(412)\ \gamma_2\gamma_1\alpha_1 : \Gamma \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\lambda_2\lambda_1\beta_1$$
$$(612)\ \alpha_2 : \Gamma \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\beta_2$$

Now via Proposition 5, we can use (71), (11), (31) and (51) and derive a contradiction from the following.

$$(\beta \cdot w)\lambda_2\lambda_1\beta_2 \not\sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\lambda_2\lambda_1\beta_2$$
$$(\beta \cdot w)\beta_1 \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\beta_1$$
$$(\beta \cdot w)\lambda_2\lambda_1\beta_1 \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\lambda_2\lambda_1\beta_1$$
$$(\beta \cdot w)\beta_2 \sim_{n-|ww_1|} (\beta \cdot ww_1)\lambda_1\beta_2$$

All remaining cases are similar. □

We use Proposition 10 to identify when a goal is final via extensions.

Definition 7. *Assume a family of not necessarily distinct goals*

$g(1)\ \alpha\alpha_1 : \Gamma \doteq \beta\beta_1$	$h(1)\ \alpha\alpha_2 : \Gamma \doteq \beta\beta_2$
$g(2)\ \alpha\gamma_1\alpha_1 : \Gamma \doteq \beta\lambda_1\beta_1$	$h(2)\ \alpha\gamma_1\alpha_2 : \Gamma \doteq \beta\lambda_1\beta_2$
$g(3)\ \alpha\gamma_2\gamma_1\alpha_1 : \Gamma \doteq \beta\lambda_2\lambda_1\beta_1$	$h(3)\ \alpha\gamma_2\gamma_1\alpha_2 : \Gamma \doteq \beta\lambda_2\lambda_1\beta_2$
$g(4)\ \alpha\gamma_1\gamma_2\gamma_1\alpha_1 : \Gamma \doteq \beta\lambda_1\lambda_2\lambda_1\beta_1$	$h(4)\ \alpha\gamma_1\gamma_2\gamma_1\alpha_2 : \Gamma \doteq \beta\lambda_1\lambda_2\lambda_1\beta_2$

(or their symmetric versions) in a branch of a proof tree involving extensions (γ_1, λ_1), (γ_2, λ_2). *If* $h(4)$ *is below all the* $g(i)$*'s and the other* $h(i)$*'s, is distinct from* $g(4)$ *and* $h(3)$ *and there is an application of UNF between* $h(3)$ *and* $h(4)$ *then* $h(4)$ *is a* successful *final goal.*

Example 6. Consider the following goals in the initial part of the potentially infinite branch of Example 5.

$g(1)$	$XZ \doteq AW : W$
$g(2) = h(1)$	$XXZ \doteq AAW : W$
$g(3) = h(2)$	$XXXZ \doteq AAAW : W$
$g(4) = h(3)$	$XXXXZ \doteq AAAAW : W$
$h(4)$	$XXXXXZ \doteq AAAAAW : W$

Here $\beta = X$ and $\alpha = Y$ and the extensions are (X, A). There is at least one application of UNF between $h(3)$ and $h(4)$ in the proof tree. Consequently, the branch stops at the final goal $XXXXXZ \doteq AAAAAW : W$. □

Example 7. If there is a repeat goal in the proof tree

$$\frac{(g)\ \alpha : \Gamma \doteq \beta}{\begin{array}{c}\vdots\end{array}}$$
$$(h)\ \alpha : \Gamma \doteq \beta$$

with an application of UNF inbetween, then h is final. Here $g(1) - g(4)$ and $h(1) - h(3)$ is the goal g with extension (ϵ, ϵ) and $\alpha_1 = \alpha_2 = \beta_1 = \epsilon$. □

Definition 8. *A* successful *tableau for $\Gamma \doteq \Delta$ is a finite proof tree with root $\Gamma \doteq \Delta$ and all of whose leaves are successful final goals. Otherwise a tableau is* unsuccessful*: that is, if it is not a finite proof tree or if it contains an unsuccesssful final goal.*

We now come to the main results, which show decidability of language equivalence for special 2SG. The decision procedure is to build the tableau with root $\Gamma \doteq \Delta$ breadth first starting with leftmost non-final goals. If an unsuccessful final goal is met then the procedure terminates with a finite unsuccessful tableau.

Theorem 2. *There is a unique finite tableau for goal $\Gamma \doteq \Delta$.*

Proof. Uniqueness is clear because rules are applied in a particular order. The important part of the proof is to show finiteness. Initially, we have $\Gamma \doteq \Delta$. The DEC rules are applied first in the order DEC(L) then DEC(R). Clearly, in the application of a DEC rule if w is the smallest word such that $\alpha \xrightarrow{w} \Lambda$ then there is no push transition in this sequence of transitions. If $(\beta \cdot w)$ involves a push transition then the tableau construction will terminate with an unsuccesful final goal. Assume the rule is DEC(L), so $\alpha : (\beta \cdot w) \doteq \beta$. Consequently, $w = w_1 a w_2$ and $\alpha \cdot w_1 = \alpha_1$ and $\beta \cdot w_1 = \beta_1$ and $\beta_1 \cdot a = \beta_1 : \beta_1$. The subgoal $\alpha_1 : (\beta \cdot w) \doteq \beta_1$ is, therefore, an unsuccessful final goal. There can not be an infinite sequence of consecutive applications of DEC as each application decreases the the number of stacks in both subgoals. Consequently, non-final subgoals to which DEC and SIMP do not apply have the form $X\alpha : \Gamma \doteq \beta$ or $\beta \doteq X\alpha : \Gamma$. First, consider the case of an application of UNF where $X \xrightarrow{a}$ push. If $\Gamma \neq \Lambda$ then the goal $(X\alpha : \Gamma) \cdot a \doteq (\beta \cdot a)$ is an unsuccessful final goal (and similarly for its symmetric version). If $\Gamma = \Lambda$, then $X\alpha \cdot a \doteq \beta \cdot a$ is $X\alpha : X\alpha \doteq \beta : \beta$ and by DEC(L) this reduces to the two occurrences of successful final goals $X\alpha \doteq \beta$ by Example 7. Consequently, without loss of generality, assume there is an infinite subbranch of goals of the form $\alpha_i : \Gamma \doteq \beta_i$, $i \geq 0$ involving applications of UNF and SIMP only. We show that there is a successful final goal. The size of the goals (that is the sum, $|\alpha_i| + |\beta_i|$) must be eventually increasing, otherwise a repeat goal

occurs ensuring a successful final goal. Now we examine the first "low point" with respect to the left stack α_i: $\alpha_i = X\alpha$ is a low point if for all $j \geq i$, $\alpha_j = \alpha'_j\alpha$. With respect to the left side we will find infinitely many repeating patterns of the form $Z\alpha'$, $Z\alpha'_1\alpha'$, $Z\alpha'_2\alpha'_1\alpha'$ and $Z\alpha'_1\alpha'_2\alpha'_1\alpha'$ where α'_1 or α'_2 can be ϵ. Now we consider the right hand stacks with respect to these repeating patterns. Clearly, we will also eventually find repeating patterns too, and consequently a successful final goal. □

Theorem 3. *The tableau for $\Gamma \doteq \Delta$ is successful iff $\Gamma \sim \Delta$.*

Proof. Suppose there is a successful tableau for $\Gamma \doteq \Delta$ but $\Gamma \not\sim \Delta$. By Theorem 2 this tableau is finite. There is a least approximant n such that $\Gamma \not\sim_n \Delta$. We construct an offending path of false goals through the tableau within which the approximant indices decrease whenever UNF is applied (by Proposition 7). The other rules preserve falsity indices. Because the tableau is finite and successful this means that the path of false goals must conclude with a final goal. But this is impossible. Clearly it is not possible to reach a final goal of the form $\Gamma \doteq \Gamma$. Moreover it is not possible to reach a final goal which is a result of extensions because of Proposition 10.

For the other direction, one just builds the tableau for $\Gamma \doteq \Delta$. By Propositions 6, 8 and 9, the applications of rules preserve truth. Therefore it is not possible to reach an unsuccessful final goal, and by Theorem 2 the tableau for $\Gamma \doteq \Delta$ is finite, and therefore successful. □

More work needs to be done to ascertain the exact complexity bound of the decision procedure.

Acknowledgements: Many thanks to Luke Ong for imparting his incisive understanding of higher-order schemes and the safety restriction, to Wong Kariento for a copy of the Maslov paper and to the referees for suggesting improvements.

References

1. Aehlig, K., De Miranda, J., and Ong C.-H. L. (2005) Safety is not a restriction at level 2 for string languages. *Lecture Notes in Computer Science*, **3411**, 490-511.
2. Aho, A. (1968). Indexed grammars–an extension of context-free grammars. *Journal of ACM*, **15**, 647-671.
3. Aho, A. (1969). Nested stack automata. *Journal of ACM*, **16**, 383-406.
4. Baeten, J., Bergstra, J., and Klop, J. (1993). Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of ACM*, **40**, 653-682.
5. Blumensath, A. (2004). A pumping lemma for higher-order pushdown automata. Preprint.
6. Bouajjani, A. and Meyer, A. (2004). Symbolic reachability analysis of higher-order context processes. *Lecture Notes in Computer Science*, **3328**, 135-147.
7. Burkart, O., Caucal, D. Moller, F., and Steffen, B. (2001). Verification on infinite structures. In *Handbook of Process Algebra*, ed. Bergstra, J., Ponse, A., and Smolka, S., 545-623, North-Holland.

8. Cachat, T. (2003). Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. *Lecture Notes in Computer Science*, **2719**, 556-569.
9. Caucal, D. (2002). On infinite terms having a decidable monadic theory. *Lecture Notes in Computer Science*, **2420**, 165-176.
10. Courcelle, B. (1978). A representation of trees by languages I and II. *Theoretical Computer Science*, **6**, 255-279 and **7**, 25-55.
11. Damm, W. (1982). The IO- and OI-hierarchy. *Theoretical Computer Science*, **25**, 95-169.
12. Damm, W., and Goerdt, A. (1986). An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, **71**, 1-32.
13. Engelfriet, J. (1991). Iterated stack automata and complexity classes. *Information and Computation*, **95**, 21-75.
14. Fischer, M. (1968). Grammars with macro-like productions. *Procs. 9th Annual IEEE Symposium on Switching and Automata Theory*, 131-142.
15. Freidman, E. (1976). The inclusion problem for simple languages. *Theoretical Computer Science*, **1**, 297-316.
16. Gilman, R. (1996). A shrinking lemma for indexed languages. *Theoretical Computer Science*, **163**, 277-281.
17. Jančar, P., and Srba, J. (2006). Undecidability results for bisimilarity on prefix rewrite systems. *Lecture Notes in Computer Science*, **3921**, 277-291.
18. Knapik, T., Niwiński, D., and Urzyczyn, P. (2002). Higher-order pushdown trees are easy. *Lecture Notes in Computer Science*, **2303**, 205-222.
19. Korenjak, A and Hopcroft, J. (1966). Simple deterministic languages. *Procs. 7th Annual IEEE Symposium on Switching and Automata Theory*, 36-46.
20. Maslov, A. (1976). Multilevel stack automata. *Problems of Information Transmission*, **12**, 38-43.
21. Ong, C.-H. L. (2006) On model-checking trees generated by higher-order recursion schemes. Preprint.
22. Parchmann, R., Duske, J. and Specht, J. (1980). On deterministic indexed languages. *Information and Control*, **45**, 48-67.
23. Sénizergues, G. (2001). L(A) = L(B)? decidability results from complete formal systems. *Theoretical Computer Science*, **251**, 1-166.
24. Sénizergues, G. (2002). L(A) = L(B)? a simplified decidability proof. *Theoretical Computer Science*, **281**, 555-608.
25. Sénizergues, G. (2003). The equivalence problem for t-turn DPDA is co-NP. *Lecture Notes in Computer Science*, **2719**, 478-489.
26. Sénizergues, G. (2005). The bisimulation problem for equational graphs of finite out-degree. *SIAM Journal of Computing*, **34**, 1025-1106.
27. Stirling, C. (2001). Decidability of DPDA equivalence. *Theoretical Computer Science*, **255**, 1-31.
28. Stirling, C. (2002) Deciding DPDA equivalence is primitive recursive. *Lecture Notes in Computer Science*, **2380**, 821-832.
29. Vijay-Shanker, K. and Weir, D. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, **27**, 511-546.